The K-Nearest Neighbor Attractor-based Neural Network and the Optimal Linear Discriminatory Filter Classifier

Gerald J. Dobeck
Naval Surface Warfare Center Panama City
Panama City, Florida 32407

Abstract — **The K-Nearest Neighbor Attractor-based Neural Network and the Optimal Linear Discriminatory Filter Classifier are feature-based classifiers that are trained via supervised learning using a training set of feature vectors. They were developed by the author and successfully used in several applications where they were "and-ed." Results using these classifiers have been published, but comprehensive descriptions of them have not appeared in the literature. This paper presents their detailed descriptions.**

I. INTRODUCTION

The K-Nearest Neighbor Attractor-based Neural Network (KNNANN) is a probabilistic-type of neural network. It was developed based on concepts taken from the Probabilistic Neural Network (PNN [1]), Reduced Coulomb Energy neural network (RCE [2]), and the K-Nearest Neighbor classifier (KNN [2]). It estimates the joint posteriori probability, *Prob(class | feature vector)*, from training data. Specifically, the KNNANN training process produces a joint posteriori histogram of the training set under the constraint of equally likely classes even if the training set is unbalanced (i.e., each class has a different number of training samples.)

The Optimal Linear Discriminatory Filter Classifier (ODFC) is a classifier that uses a bank of linear classifiers that are referred to as linear discriminatory filters. The coefficients of the linear classifiers are found by solving a generalized eigenvalue problem. There are generic similarities between ODFC optimization approach and the one described in [3]. The outputs of the linear classifiers are combined to produce an overall statistic that is used to determine class membership.

These two classifiers are complementary in how they construct decision boundaries in feature space. The KNNANN uses multiple hyperspheres (also referred to as attractors) to define class boundaries, and the ODFC uses multiple hyperplanes. As such they have been combined by the author using various "anding" schemes, which resulted in overall performance that was much better than the performance of either one alone.

A desirable attribute of both classifiers is that their training processes are not iterative and are computationally very fast. This permits the designer to efficiently evaluate many feature subsets from large set of candidate features and select the subset that performs "best" or "good enough," thereby overcoming Bellman's well-known curse of dimensionality [4].

These classifiers were developed by the author several years ago and used in several applications [5-8]. In section III the KNNANN is described in detail, and in section IV the ODFC is described.

II. FEATURE VECTOR NORMALIZATION

Both classifiers perform better if feature vectors are normalized. The normalization process is described in this section. Define

$$x(k) = k\text{-th feature vector from the training set.} \quad (1)$$
$$class(k) = \text{class of } x(k) \in \{1, 2, \ldots, N_c\}$$

where

N_c = Number of classes
$k = 1, 2, \ldots N$
N = Number of training feature vectors
$M(i)$ = Number of training vectors belonging to class i

Let $x_j(k)$ denote the j-th feature component of the k-th feature vector. The normalized feature component is given by

$$x_j(k; \text{"normalized"}) = [\, (x_j(k; \text{"original"}) - bias(j) \,] \,/\, scale(j) \quad (2)$$

where

$$w(k) = \text{sample weights} = 1 \,/\, M(class(k)) \quad \text{for } k = 1, 2, \ldots N \quad (3)$$

$$W = \sum_{k=1}^{N} w(k) \quad (4)$$

$$\text{bias}(j) = (1/W) \sum_{k=1}^{N} w(k)\, x_j(k) \quad (5)$$

$$\text{scale}(j) = (1/W) \sum_{k=1}^{N} w(k)\, |\, x_j(k) - \text{bias}(j) \,| \quad (6)$$

Regarding this normalization process: (1) scale and bias are computed using only the training data and must be saved for application to any new feature vector, (2) the use of $w(k)$ imposes an equally likely class probability assumption. This prevents any class with a much greater number of samples than the other classes from dominating the calculations. For the remainder of the paper, the feature vectors $x(k)$ are assumed to be normalized.

As an aside, in some applications it is sometimes beneficial to base the determination of the bias and scale on samples from one key class (or a few key classes) rather than all classes as presented here.

1-4244-0115-1/06/$20.00 ©2006 IEEE

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 SEP 2006	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
The K-Nearest Neighbor Attractor-based Neural Network and the Optimal Linear Discriminatory Filter Classifier	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Surface Warfare Center Panama City Panama City, Florida 32407**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002006. Proceedings of the MTS/IEEE OCEANS 2006 Boston Conference and Exhibition Held in Boston, Massachusetts on September 15-21, 2006, The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**6**	

III. KNNANN

The first stage of the KNNANN Training algorithm determines the number of attractors (hyperspheres) that will contain all the feature vectors from training set, their center locations in feature space, and their associated radii. Let

$c(L)$ = center of attractor L (7)
$r(L)$ = radius of attractor L

The second stage of the KNNANN Training algorithm determines $p(i, L)$, the estimated probability that an object belongs to class i given that its feature vector is contained in attractor L.

The KNNANN Training algorithm uses some selectable design parameters: α, β, and $\gamma(i)$:

α = nearest neighbor factor (8)
where
$0 < \alpha < 1$
Recommend: $\alpha = 0.5$

β = radius dither parameter (9)
where
$0 \le \beta < 0.5$
Recommend: $\beta = 0.005$

$\gamma(i)$ = removal factor for $i = 1, 2, \ldots N_c$ (10)
where
$0 \le \gamma(i) \le 1$
Recommend: $\gamma(i) = 0.0$ if $M(i)$ is small
$= 0.5$ if $M(i)$ is of modest size
$= 1.0$ if $M(i)$ is large

The selection and usage of these parameters will be discussed later, but first the training algorithm is presented.

The KNNANN uses a vector norm, which will be denoted as $norm(u)$ for the norm of vector u. The standard L_2 norm is recommended. Other norms can be used to derive different variants of the KNNANN.

KNNANN Training Algorithm

1. Initialization:
   Set $L = 0$
   $k_{ref} = 0$
   $flag(k) = 0$ for $k = 1, 2, \ldots N$
2. Set $k_{ref} = k_{ref} + 1$
   If $k_{ref} > N$
   Then: Go to step 12
   Else: If $flag(k_{ref}) = 0$
   Then: Go to step 3
   Else: Go to step 2
3. Set $L = L+1$
   $c(L)$ = center of attractor
   $= x(k_{ref})$
4. Set $d(k) = norm(x(k) - c(L))$ and
   $v(k) = [M(class(k))]^{-1/2}$ for $k = 1, 2, \ldots N$.
5. Sort $d(k)$ from smallest to largest. Let k_m be the sorted index list; i.e., $d(k_1) \le d(k_2) \le \ldots \le d(k_N)$.
6. Find the smallest M such that
   $\sum_{m=1}^{M} v(k_m) \ge \alpha$
7. Set $r(L)$ = radius of the attractor
   $= (1+\beta)\, d(k_M)$
8. Set $Q = 0$
   $q(i) = 0$ for $i = 1, 2, \ldots N_c$.
9. For $m = 1, 2, \ldots N$
   Set $k = k_m$
   If $d(k) \le r(L)$,
   Then: $w = 1 / M(class(k))$
   $Q = Q + w$
   $q(class(k)) = q(class(k)) + w$
   If $d(k) \le \gamma(class(k))\, r(L)$
   Then: $flag(k) = 1$
   Else: continue
   Else: Go to step 10.
   End of For m loop
10. For $i = 1, 2, \ldots N_c$
    $p(i, L) = q(i) / Q$
    End of For i loop
11. Return to step 2
12. Training is complete.

When training is completed, L attractors have been identified with radii $r(j)$, centers $c(j)$, and estimated class probabilities $p(i, j)$ for $i = 1, 2, \ldots N_c$ and $j = 1, 2, \ldots L$. $p(i, j)$ is an estimate of posteriori probability, $Prob(class\ i \mid x = c(j))$, under the assumption of equally likely class probabilities. As desired, note that

$$\sum_{i=1}^{N_c} p(i, j) = 1 \quad (11)$$

The following algorithm describes how this information is used to determine the class of an arbitrary feature vector, x. The evaluation algorithm uses a single selectable design parameter, h.

h = attractor radius expansion factor (12)
where
$0 \le h$
Recommend: $h = 1.5$

The selection and usage of parameters h will be discussed later, but first the evaluation algorithm is presented.

KNNANN Evaluation algorithm

1. Initialize: $Q = 0$,
   $q(i) = 0$ for $i = 1, 2, \ldots N_c$

2. For $j = 1, 2, \ldots L$
   $d = norm(x - c(j))$
   If $d \le h\, r(j)$
   Then: $w = 1 / (d + 0.000001)$
   $Q = Q + w$
   For $i = 1, 2, \ldots N_c$
   $q(i) = q(i) + w\, p(i, j)$
   End of For i loop
   Else: continue
   End of For j loop
3. If $Q > 0$
   Then: For $i = 1, 2, \ldots N_c$
   $z(i \mid x) = q(i) / Q$
   End of For i loop
   Else: For $i = 1, 2, \ldots N_c$
   $z(i \mid x) = 0$
   End of For i loop

If $Q > 0$ upon completion of these three steps, $z(i \mid x)$ is an estimate of *Prob (class i | x)*. It is easy to show that the sum of $z(i \mid x)$ over i is one. If $Q = 0$, then x did not belong to the neighborhood of any attractor. In this case, it is assumed that x is statistically dissimilar to the training set. Therefore there is no basis for assigning a class; the class is declared "unknown."

Let

$$I_{max} = \underset{i}{argmax}(z(i \mid x)) \tag{13}$$

$$I_{ref} = \underset{i \neq I_{max}}{argmax}(z(i \mid x))$$

T = a designer selectable classification threshold.

The classification rule is given by:

If $z(I_{max} \mid x) \geq T\, z(I_{ref} \mid x)$, then class = I_{max}; otherwise the class is considered uncertain.

For a two-class problem (e.g., class *1* = target and class *2* = nontarget), the classification rule is given by:

If $z(1 \mid x) > T\, z(2 \mid x)$, then class = *1*; otherwise class = *2*.

T can be selected to adjust the operating point for probability of classification versus probability of false alarm. Furthermore, if a sufficiently large set of feature vectors is available with their associated class labels, the Receiver Operator Characteristic curve can be estimated by varying T. (So as not to be fooled by possible overtraining, one should use feature vectors that were not used in the training to generate the ROC curve.)

Other variants of interpolation can be used in the KNN Evaluation algorithm by simply changing the interpolation weight, w, in step 2. For example, one could use

$$w = exp(-a \mid d / r(j) \mid^{b}) \tag{14}$$

for appropriate choices of parameters $a > 0$ and $b > 0$.

KNNANN design parameters

The KNNANN training algorithm estimates a joint histogram of the training data where each attractor represents a volume in feature space in which one counts the number of training samples that belong to each class. The KNNANN counting process, which is used to size the attractor and determine the sample probabilities associated with it, performs two key functions: (1) it balances the training data when the classes may have a disparate number of samples, and (2) it sizes each attractor so that the number of samples it contains is compatible with the size of the training set. To illustrate, suppose one is creating a histogram from M samples where the histogram volumes (attractors) that cover feature space can vary in size. Assume each volume contains, on the average, M_s samples and that M_v volumes contain all M samples. If the volumes are disjoint, then $M = M_s M_v$. The question is how many volumes M_v should one strive to have and how many samples M_s per volume. We would like M_s to be large so that the sample probabilities for each volume can be accurately estimated. However, we would also like M_v to be large so that the joint histogram has many small attractors that have sufficient resolution to accurately approximate the continuous joint probability density over all of feature space. A natural way to balance M_v versus M_s is by maximizing

$$J(M_s, M_v) = min(\alpha^2 M_v, M_s) \tag{15}$$

under the constraint that $M_v M_s = M$ where parameter α is a tradeoff weight. The solution is given by

$$M_s = \alpha\, M^{1/2} \tag{16}$$

and

$$M_v = (1/\alpha)\, M^{1/2}$$

(We are ignoring the fact the solution is, in general, not an exact integer because it is not critical to this discussion.)

Note that tradeoff weight, α, is the "nearest neighbor factor" that was defined earlier in Eq 8.

Furthermore note that $M_s(M) = \alpha\, M^{1/2}$ has the asymptotic properties that

$$\lim_{M \to \infty} M_s(M) = \infty \tag{17}$$

$$\lim_{M \to \infty} M_s(M) / M = 0$$

These properties imply that the joint histogram will approach the true joint probability density as M approaches infinity.

However we must also keep in mind that the training set may not be balanced (i.e., some classes may have more training samples than others). We do not want the training process to favor one class over the others simply because it has more samples. Accordingly, each sample is weighted by

$$[\, M(class(k)\,)\,]^{-1/2}. \quad (18)$$

.

Then the radius of an attractor is given as the smallest radius such that the sum of the weights of the samples contained in the attractor is greater than or equal to parameter α. This is implemented in steps 4, 5, and 6 of the KNNANN Training algorithm. For the special case where the attractor contains only class i, one can show that the number of samples in the attractor is the smallest integer that is greater than or equal to

$$\alpha\,[\, M(i)\,]^{1/2}. \quad (19)$$

To determine the sample probabilities within an attractor and account for unbalanced training data, each sample is weighted by

$$1 / M(i). \quad (20)$$

By using these weights in a counting procedure to determine sample probabilities, all classes appear to have the same number of training samples (i.e., classes appear equally likely). This is implemented in steps 8, 9, and 10 of the KNNANN training algorithm. To illustrate, consider the special case where the number of samples from class i that belongs to attractor L is $P_0\ M(i)$ for all i where P_0 is any fractional constant. Then $p(i, L)$ will equal $1/N_c$ for all i.

The parameter $\gamma(i)$ is used to control the amount of attractor overlap. It also influences the number of attractors that will be determined. See the second "if-statement" in step 9 of the KNNANN Training algorithm. Note that any of the feature vectors, which have not been flagged in step 9, have the potential to be selected as an attractor center. If a feature vector belonging to class i is within radius $\gamma(i)\ r(L)$ of the center of attractor L, it is flagged so it cannot be selected as a new attractor center in subsequent passes through the algorithm. Consequently, if $\gamma(i) = 0$, then every feature vector belonging to class i will be selected as an attractor center. For $\gamma(i) = 0$, observe that attractors, whose centers have been selected from class i, have the potential for large overlap because a new attractor center is permitted to be very near an existing attractor center. In contrast, for $\gamma(i) = 1$, observe that attractors, whose centers have been selected from class i, have a more limited potential to overlap because a new attractor center must be at least a full radius away from all existing attractor centers.

To reduce the number of attractors that must be stored and searched during evaluation, it is recommended to use $0.5 < \gamma(i) < 1$ when $M(i)$ is large. To make maximum use of class i data when $M(i)$ is small, it is recommended that $\gamma(i) = 0$.

Note when $\gamma(i) = 1$ for all i, a minimal number of attractors is generated; and when $\gamma(i) = 0$ for all i, the largest number of attractors is generated. In practice, $\gamma(i)$ is selected by trial and error. Typically, for networks trained with *1000* to *4000* feature vectors, $\gamma(i)$ is selected so that network will have between *250* and *500* attractors. Note that training time decreases substantially as $\gamma(i)$ becomes large because more vectors are flagged for removal as attractors are selected; this is especially useful for large training sets. It is instructive to note that the number of passes through the training algorithm (i.e., the number of times steps 3 - 11 are executed) is finite and cannot exceed N. More specifically, if $\gamma(i)$ are near *0*, the number of passes is of order N; if $\gamma(i)$ are near *1*, the number of passes is of order $N^{1/2}$.

The training algorithm can be made faster by noting that only a partial sort is required in step 5.

Because the attractors are likely to overlap, an arbitrary feature vector is often contained in more than one attractor. This is beneficial because it allows one to obtain a better estimate of the class probabilities by interpolating the sample probabilities from each attractor. This interpolation is done in the KNNANN Evaluation algorithm. It is a weighted average of the attractors' sample probabilities where the weights favor the attractors whose centers are nearest the feature vector that is being classified.

The design parameter h, the radius expansion factor that is used in the KNNANN Evaluation algorithm (Eq 12), controls which attractors to associate with the feature vector that is being classified. If the feature vector is not contained in any attractor, it is considered as statistically dissimilar to the training set and, consequently, the class is declared as unknown. However, because the design of the network was based on a finite training set, it is statistically possible for a new feature vector not to be contained in any attractor but near the perimeter of some of them. The expansion factor is used to increase an attractor's radius of influence so that it will capture these nearby feature vectors. In addition it permits more attractors to be involved in the interpolation process.

IV. ODFC

The ODFC uses the same notation from Eqs 1 and 2. However the feature vector is augmented with an additional component, which is set to unity. Namely,

$$x_{n+1}(k) = 1 \quad (21)$$

As will be obvious later, the linear coefficient that multiplies this component acts as an offset bias and allows the hyperplane (as observed in the original n-dimensional feature space) to optimally shift and better align with class boundaries. For the remaining ODFC sections, x is assumed to be the augmented $(n+1)$-dimensional feature vector.

Define a bank of coefficient vectors as

$$\begin{aligned} c(i, j) &= \text{the } j\text{-th coefficient vector associated with class } i \\ &= (n+1)\text{-dimensional column vector} \end{aligned} \quad (22)$$

where

$$i = 1, 2, \ldots N_c \text{ and } j = 1, 2, \ldots N_f(i) \quad (23)$$

$N_f(i)$ = number of coefficient vectors associated with class i

The first n components of $c(i, j)$ define the normal vector of a hyperplane in the original n-dimensional feature space. The $n+1$ component of $c(i, j)$ accounts for the shift of the

hyperplane relative to the origin of the *n*-dimensional feature space.

Define the output of the filters as the projection of *x* along *c(i, j)*; namely,

$$s(x, c(i, j)) = c'(i, j)\, x \tag{24}$$

where $'$ is the matrix-vector transpose operator.

The training goal is to determine *c(i, j)* such that, on the average, $s^2(x(k), c(i, j))$ is large for *x(k)* belonging to class *i* and is small for *x(k)* not belonging to class *i* for *j = 1, 2, ... $N_f(i)$*.

Define the correlation matrix

$$A(i) = (1/M(i)) \sum_{x(k) \in class\ i} x(k)\, x'(k) \tag{25}$$

Then

$$(1/M(i)) \sum_{x(k) \in class\ i} s^2(x(k), c) = c'\, A(i)\, c. \tag{26}$$

For clarity, the indices on *c(i, j)* have been dropped. The solution is found by maximizing *J(c)* with respect to *c* where

$$J(c) = \frac{c'\, A(i)\, c}{c'\, B(i)\, c} \tag{27}$$

and

$$B(i) = (1 / (N_c - 1)) \sum_{m \neq i} A(m) \tag{28}$$

One notes that the numerator of *J(c)* is the sample expectation of $s^2(x, c)$ over the training set given that *x* belongs to class *i*. And the denominator of *J(c)* is the sample expectation of $s^2(x, c)$ over the training set given that *x* does not belongs to class *i* under the assumption of equally likely classes. That is

$$J(c) = \frac{E_s [\, s^2(x, c) \mid x \in class\ i \,]}{E_s [\, s^2(x, c) \mid x \notin class\ i \,]} \tag{29}$$

where $E_s[.]$ denotes the sample expectation (or sample mean).

One recognizes *J(c)* as the Raleigh quotient implying the solution is found by solving the generalized eigenvalue problem:

$$A(i)\, v - \lambda\, B(i)\, v = 0 \tag{30}$$

If the eigenvector *v* and eigenvalue λ is a solution to Eq 30, then

$$\lambda = \frac{v'\, A(i)\, v}{v'\, B(i)\, v} = \frac{E_s [\, s^2(x, v) \mid x \in class\ i \,]}{E_s [\, s^2(x, v) \mid x \notin class\ i \,]} \tag{31}$$

This implies that *c(i, 1), c(i, 2), ... c(i, N_f (i))* should be selected to be all the eigenvectors whose eigenvalues are large relative to 1.

If *c(i, j)* is scaled as follows,

$$c(i, j) = v / (v'\, B(i)\, v)^{1/2} \tag{32}$$

it is easy to show that,

$$E_s[\, s^2(x, c(i, j)) \,] = \lambda \quad \text{for } x \in \text{class } i \tag{33}$$

$$E_s[\, s^2(x, c(i, j)) \,] = 1 \quad \text{for } x \notin \text{class } i$$

Selection of eigenvectors for *c(i, j)*

The procedure for determining which eigenvectors to select uses two selectable design parameters, *H* and *K*. All eigenvectors are selected whose eigenvalues are greater than *H*. However, if more than *K* eigenvectors have been selected, only the eigenvectors corresponding to the *K* largest eigenvalues are kept. Because *A(i)* and *B(i)* are *(n+1)* x *(n+1)* matrices, the maximum number of eigenvectors is *n+1*. Consequently $N_f(i)$ obeys $0 \le N_f(i) \le K \le n+1$. In our applications the choices of *H = 2.0* and *K = min(5, n+1)* have performed well. If no eigenvalues are greater than *H* (i.e., *Nf(i) = 0*), then one selects the eigenvector corresponding to the maximum eigenvalue (so $N_f(i)$ will be at least *1*). When the eigenvalues are not large, the implication is that the features are not providing good discrimination and classifier performance is generally poor.

Finally, the selected eigenvectors are scaled as described above in Eq 32 and stored in *c(i, j)*.

The ODFC training process, described by Eqs 25 to 32, is repeated for *i = 1, 2, ... N_c*.

Conditioning matrix *A(i)*

If the number of class samples is sufficiently large and the features are linearly independent, then *A(i)* and *B(i)* are real symmetric positive definite matrices. Fast and accurate computer routines for solving the generalized eigenvalue problems having such matrices are readily available in many linear algebra software packages (e.g., IMSL and MATLAB). To increase robustness to random errors on the feature vector and mitigate the possibility of ill-conditioning when solving the generalized eigenvalue problem, the first *n* diagonal elements of *A(i)* are multiplied by factor *(1 + ε)* prior to generating *B(i)*. One should try several values for ε and choose the one that gives the best results with a test set of feature vectors. In many of our applications the choice of ε = 0.0025 has performed well.

Summary of the ODFC Training Process

The ODFC Training process is summarized as follows. Initially augment each training feature vector with one additional component and set it equal to 1.

For *i = 1, 2, ...N_c*

1. Compute *A(i)* from Eq 25.

2. As described above in subsection "Conditioning matrix $A(i)$,"condition the diagonal elements of $A(i)$.
3. Calculate $B(i)$ by Eq 28.
4. Solve for the eigenvectors and eigenvalues of Eq 30.
5. As described above in subsection "Selection of eigenvectors for $c(i, j)$," choose parameters H and K and use them to select the appropriate subset of $N_f(i)$ eigenvectors.
6. According to Eq 32, scale the selected eigenvectors and save them in $c(i, j)$ for $j = 1, 2, \ldots N_f(i)$.

End of For i loop

ODFC Evaluation process

Using $c(i, j)$ for $i = 1, 2, \ldots N_c$ and $j = 1, 2, \ldots N_f(i)$, the class of a new feature vector x is determined as follows. Define

$$z(i \mid x) = \max_j (\mid c'(i, j)\, x \mid) \quad (34)$$

$$I_{max} = \operatorname*{argmax}_i (z(i \mid x))$$

$$I_{ref} = \operatorname*{argmax}_{i \neq I_{max}} (z(i \mid x))$$

T = a designer selectable classification threshold.

The classification rule is given by:

If $z(I_{max} \mid x) \geq T\, z(I_{ref} \mid x)$, then class = I_{max}; otherwise the class is considered uncertain.

For a two-class problem (e.g., class *1* = target and class *2* = nontarget), a classification threshold, T, can be introduced to adjust the probability of classification relative to the probability of false alarm. The classification rule is given by:

If $z(1 \mid x) > T\, z(2 \mid x)$, then class = *1*; otherwise class = *2*.

V. CONCLUSION

This paper presented detailed descriptions of the K-Nearest Neighbor Attractor-based Neural Network (KNNANN) and the Optimal Linear Discriminatory Filter Classifier. Both are feature-based classifiers whose training is based on supervised learning.

The KNNANN is a probabilistic-based neural network that constructs a joint histogram, which approximates the joint posteriori class probability density function. Some novel attributes of the KNNANN are:

(1) A training procedure that balances the size of the attractors so that they are small enough to resolve the underlying probability density but large enough to hold a sufficient number of training samples to accurately estimate the class probabilities.

(2) A compensation method for unbalanced training data sets (i.e., data sets where the number of samples in each class differ substantially).

(3) An interpolation method that estimates class probabilities by combining sample probabilities of several overlapping attractors.

The ODFC employs multiple linear classifiers that are combined to determine the class of a feature vector. Coefficients of the linear filters are derived by solving a generalized eigenvalue problem.

Because the KNNANN uses hyperspheres to define class boundaries in feature space and the ODFC uses hyperplanes, the two classifiers are complementary and have performed well when "anded."

The training algorithms for both classifiers are not iterative and computationally very fast. This allows one to use the classifiers to evaluate a large number of feature subsets taken from a large set of candidate features and select the subset that performs the "best" or "good enough," thereby overcoming Bellman's well-known curse of dimensionality.

ACKNOWLEDGMENT

This work was funded by the Office of Naval Research, Dr. Tom Swean, ONR 321OE.

REFERENCES

[1] D. F. Specht, "Probabilistic Neural Networks," Neural Networks, Vol. 3, pp 109-118, 1990.

[2] R. Duda, P. Hart, D. Stork, Pattern Classification, John Wiley and Sons, Inc., 2001.

[3] A. Mahalanobis, R. Muise, S. Stanfill, A. Van Nevel, "Design and application of quadratic correlation filters for target detection," IEEE Transaction on Aeorspace and Electronc Systems, Volume 40, Issue 3, pp 837-850, July 2004.

[4] G. Dobeck, J. T. Cobb, "False alarm reduction by the And-ing of multiple multivariate Gaussian classifiers," Proceedings of SPIE'03, Vol. 5089, 45-57, Orlando Florida, 21-25 April 2003.

[5] G. Dobeck, "Robust score-based feature vectors for Algorithm Fusion," Proceedings of SPIE'04, Vol. 5415, pp. 304-314, Orlando, FL, 12-16 April 2004.

[6] G. Dobeck, "Algorithm fusion for automated sea mine detection and classification," Proceedings of Oceans 2001, Vol. I, pp. 130-134, Nov. 2001.

[7] G. Dobeck, "Fusing sonar images for mine detection and classification," Proceedings of SPIE'99, Vol. 3710, pp. 602-614, Orlando, Florida, 5-9 April 1999.

[8] J. C. Hyland, G. J. Dobeck, "Sea Mine Detection and Classification Using Side-Looking Sonar," Proceedings of SPIE'95, Vol. 2496, pp. 442-453, Orlando, Florida, 17-21 April 1995.